

## ΕΡΓΑΣΙΑ ΑΝΑΚΤΗΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ

ΟΔΥΣΣΕΑΣ ΚΟΠΑΚΑΚΗΣ ΜΠΕΛΜΠΑΣ 1072653 [up1072653@upnet.gr](mailto:up1072653@upnet.gr)

ΙΩΑΝΝΗΣ ΝΙΚΟΛΑΟΥ 1072681 [up1072681@upnet.gr](mailto:up1072681@upnet.gr)

### Καταγραφή περιβάλλοντος υλοποίησης










Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η Python και συγκεκριμένα η έκδοση 3.10 μαζί με τις βιβλιοθήκες numpy, pandas, tensorflow, keras και elasticsearch (συγκεκριμένη έκδοση 7.10 κι επιλέχτηκε επειδή είναι η νεότερη έκδοση που ήταν ακόμα open source).

Η εγκατάσταση της Python είναι σχετικά απλή, πατώντας αυτό το [σύνδεσμο](#) οδηγούμαστε στη σελίδα για την εγκατάσταση της Python όπου διαλέγουμε το κατάλληλο link σύμφωνα με το λειτουργικό σύστημα που έχουμε.

Μαζί με την εγκατάσταση της Python έχουμε και τον package manager pip, οπότε μπορούμε να τρέξουμε την εντολή `pip install -r requirements.txt` ώστε να κατέβουν όλες οι απαραίτητες βιβλιοθήκες.

Επόμενο βήμα είναι να εγκαταστήσουμε την elasticsearch. Ο τρόπος που επιλέξαμε εμείς είναι μέσω του docker. Για να εγκαταστήσει κάποιος το docker ακολουθεί αυτό τον [σύνδεσμο](#). Μαζί με το docker κατεβαίνει και το docker compose.

Στο φάκελο με τον κώδικα της εργασίας υπάρχει ένα αρχείο docker-compose.yml. Πρέπει να πάμε σ' αυτό το φάκελο που βρίσκεται αυτό το αρχείο και στο terminal να τρέξουμε την εντολή `docker-compose up -d`. Με αυτή την εντολή θα δημιουργηθεί ένα container το οποίο περιέχει την elasticsearch και το kibana τα οποία μπορούμε να τα κάνουμε access από τις διευθύνσεις: <http://localhost:9200/> - Elastic Search API και <http://localhost:5601/> - Kibana Web UI interface. Κάθε φορά θα ανοίγουμε το docker desktop και θα ενεργοποιούμε τα services πατώντας το play κουμπί του elasticsearch container.

▼ 	elasticsearch	-	Exited		▶ ⋮ 🗑
	kibana aef246ecb871 	<a href="http://localhost:5601/">docker.elastic.co/kibana/kibana</a>	Exited (255)	5601:5601  9600:9600 	▶ ⋮ 🗑
	elasticsearch 53a1fb78d8d8 	<a href="http://localhost:9200/">docker.elastic.co/elasticsearch</a>	Exited (255)	9200:9200  9300:9300 	▶ ⋮ 🗑

### Ερώτημα 1

## ΕΡΩΤΗΜΑ 1

```
import pandas as pd
from elasticsearch import Elasticsearch, helpers
import csv
import numpy as np
```

✓ 1.0s

### Αποθήκευση των csv αρχείων σε dataframe

```
books = pd.read_csv('BX-Books.csv')
users = pd.read_csv("BX-Users.csv")
ratings = pd.read_csv("BX-Book-Ratings.csv")
```

✓ 1.6s

Αρχικά κάνουμε import τις απαραίτητες βιβλιοθήκες και φορτώνουμε σε dataframes τα απαραίτητα csv files μέσω της βιβλιοθήκης pandas.

```
ratings = pd.merge(ratings, users, on='uid', how='outer', indicator='Exist')
ratings = ratings[(ratings['Exist']=='both')]
ratings = ratings[['uid', 'isbn', 'rating']]
```

✓ 1.0s

```
ratings = pd.merge(ratings, books, on='isbn', how='outer', indicator='Exist')
ratings = ratings[(ratings['Exist']=='both')]
ratings = ratings[['uid', 'isbn', 'rating']]
ratings['uid'] = pd.to_numeric(ratings['uid'], errors='coerce').fillna(0).astype(int)
```

✓ 0.4s

Έπειτα κάνουμε μια προ-επεξεργασία ώστε στο dataframe ratings να μείνουν μόνο valid δεδομένα. Δηλαδή ratings από χρήστες που υπάρχουν στο dataframe users και σε βιβλία που υπάρχουν στο dataframe books. Η τελευταία εντολή `pd.to_numeric` στο δεύτερο cell χρησιμεύει ώστε να προσπαθήσουμε να κάνουμε convert το uid column σε αριθμητικές τιμές. Το argument `errors='coerce'` χρησιμοποιείται για να θέσουμε τις invalid τιμές σε NaN και στις συνέχεια τις αντικαθιστούμε με 0. Τέλος η μέθοδος `astype` χρησιμοποιείται για να μετατρέψουμε τις τιμές του συγκεκριμένου column σε ακέραιες τιμές.

## Αποθήκευση των βιβλίων στην elasticsearch

```
ENDPOINT = "http://localhost:9200/"
es = Elasticsearch(timeout=600, hosts=ENDPOINT)
es.ping()
```

✓ 0.1s

True

```
if (not es.indices.exists(index="books")):
    with open('BX-Books.csv', encoding="utf8") as f:
        reader = csv.DictReader(f)
        helpers.bulk(es, reader, index='books')
else:
    print("This index already exists")
```

✓ 0.0s

This index already exists

Στη συνέχεια έχοντας πατήσει το κουμπί run από το docker desktop ώστε να αρχίσει να εκτελείται το elasticsearch service ανοίγουμε μια σύνδεση μέσω της python στο συγκεκριμένο endpoint και ελέγχουμε αν είναι ενεργή αυτή η σύνδεση μέσω της εντολής ping. Στη συνέχεια φτιάχνουμε το index books στην elasticsearch μέσω της εντολής bulk ώστε να αποφευχθεί η ανακατασκευή του index στην περίπτωση που τα στέλναμε κάθε φορά ένα-ένα. Χρησιμοποιείται ένα if statement ώστε αν ξανά τρέξουμε το πρόγραμμα και υπάρχει ήδη το index να μην το ξαναφτιάξουμε.

Δίνεται το αναγνωριστικό του χρήστη και το αλφαριθμητικό και επιστρέφεται το 10% των βιβλίων που ταιριάζουν καλύτερα με το αλφαριθμητικό σε φθίνουσα σειρά συνυπολογίζοντας την μετρική ομοιότητας της elasticsearch, καθώς και την βαθμολογία που έχει βάλει ο χρήστης

```
while True:
    user = int(input("Write id of user: "))
    if (users['uid'].isin([user]).any()):
        break
    else:
        print(user, "is not a valid id. Please input a valid user id!")
    user_query = input("Write search term: ")
    print("uid =", user, end=', ')
    print("query =", user_query)
```

✓ 6.0s

Python

8 is not a valid id. Please input a valid user id!  
uid = 123629, query = Canada

Σ' αυτό το κομμάτι του κώδικα ζητάμε ως είσοδο να δοθεί το αναγνωριστικό του χρήστη κι ένα query με το οποίο θα ψάξουμε τα πεδία summary και title των βιβλίων που έχουμε στο index μέσα στην elasticsearch ώστε να έχουμε το καλύτερο δυνατόν ταίριασμα. Το user id ελέγχεται με το column του dataframe users ώστε να έχουμε σίγουρα valid user id.

```

good_new_ratings = ratings[(ratings["uid"] == user) & (ratings['rating'] > 0)].sort_values(by='rating', ascending=False)
goodRatedBooks = good_new_ratings['isbn'].tolist()
print(goodRatedBooks)
✓ 0.0s
['0002005018']

```

Στο δεύτερο cell βρίσκουμε από το dataframe ratings όλα τα βιβλία που έχει βαθμολογήσει ο χρήστης, διαλέγουμε αυτά που έχει βαθμολογήσει με βαθμό μεγαλύτερο του 0 (επειδή στα datasets πάρα πολλά ratings είναι ίσα με 0) και τα βάζουμε σε μια λίστα goodRatedBooks με descending order.

```

books_ex1 = es.search(index="books",
    body = {
        "query": {
            "bool": {
                "should": [
                    {
                        "multi_match": {
                            "query": user_query,
                            "fields": [
                                "summary^2",
                                "title"
                            ]
                        }
                    },
                    {
                        "bool": {
                            "should": [
                                {
                                    "match": {
                                        "isbn": {
                                            "query": book,
                                            "minimum_should_match": "1",
                                            "boost": 4.0
                                        }
                                    }
                                }
                            ]
                        }
                    }
                ]
            }
        },
        "size": 10000
    }
)
result = books_ex1["hits"]
total_hits = result['total']['value']

```

Έπειτα τρέχουμε αυτό το query στην elasticsearch. Αυτό το query κάνει αρχικά multi\_match\_search ψάχνοντας σε δύο fields summary και title με το user query δίνοντας προτεραιότητα αν τα search terms βρίσκονται στο summary field με το per field boosting operator ^2 (δίνει διπλάσια προτεραιότητα). Στη συνέχεια έχουμε μια δεύτερη παράμετρο στο query όπου ψάχνουμε μέσα στη λίστα από τα good rated books για κάθε βιβλίο αν υπάρχει στο field isbn του index και με την παράμετρο minimum\_should\_match που έχουμε θέσει 1, σημαίνει ότι τουλάχιστον 1 βιβλίο από τη λίστα πρέπει να κάνει match με το field isbn ώστε ένα document-βιβλίο από το index να θεωρηθεί match και να μας το επιστρέψει η elasticsearch. Επίσης το size τίθεται ίσο με 10,000 που είναι αρκετά μεγάλο ώστε στη συνέχεια να μπορούμε να επιλέξουμε μόνο το 10% των αποτελεσμάτων που μας επιστρέφει η elasticsearch.

Με τις δύο επόμενες εντολές επιλέγουμε από την απάντηση της elasticsearch το πεδίο hits που μας ενδιαφέρει και στη συνέχεια διαλέγουμε το πεδίο total -> value το οποίο μας επιστρέφει το συνολικό αριθμό των hits ο οποίος μπορεί να είναι μικρότερος από το size που έχουμε θέσει και τον χρειαζόμαστε στη συνέχεια για να επιστρέψουμε το 10% των αποτελεσμάτων.

```
result = books_ex1["hits"]["hits"][:int(0.1*total_hits)]
li = []
for i in result:
    res = i["_source"]
    li.append(res['isbn'])
print(li)
```

✓ 0.0s Py

['0002005018', '0968067824', '0717282201', '052164626X', '0792271521', '0870444131', '1551580004', '0771040997', '0195417836', '0770418686', '0060183284', '0919891276', '082630706X', '0618197338', '1551053020', '0395442567', '0395544092', '0618131736', '0395711797', '0395681022', '0395636272', '0395383986', '0395636280', '0618117490', '0395926890', '0864923155', '1553373405', '039593916X', '0395939186']

Τέλος παίρνουμε μια λίστα από τα αποτελέσματα πηγαίνοντας στο πεδίο hits -> hits της απάντησης της elasticsearch και κρατώντας τα 10% πιο κορυφαία. Έπειτα κάνουμε loop through στα στοιχεία της λίστας και διαλέγοντας το πεδίο "\_source" που περιέχει όλα τα απαραίτητα fields του index διαλέγουμε το field isbn και το προσθέτουμε αρχικά σε μία άδεια λίστα ώστε αυτή στη συνέχεια να γεμίσει με όλα τα κορυφαία 10% βιβλία.

Όπως βλέπουμε από τα αποτελέσματα που επέστρεψε η elasticsearch το βιβλίο που βαθμολόγησε ο χρήστης με την καλύτερη βαθμολογία (0002005018) βρίσκεται 1<sup>ο</sup> στη σειρά κατάταξης των αποτελεσμάτων που είναι σωστό αποτέλεσμα καθώς το είχε βαθμολογήσει με καλό βαθμό.

## Ερώτημα 2

Σε αυτό το ερώτημα ζητείται να ομαδοποιήσουμε τους χρήστες σε συστάδες με χρήση του αλγόριθμου k-means βάσει της χώρας στην οποία κατοικούν και της ηλικίας τους. Αυτό έγινε με τον εξής τρόπο.

## ΕΡΩΤΗΜΑ 2

```
from sklearn.cluster import KMeans
from collections import Counter
```

Python

Θα αποθηκεύσουμε τους χρήστες σε ένα νέο dataframe. Στη συνέχεια θα το τροποποιήσουμε, ώστε να εφαρμόσουμε clustering

```
new_users = pd.read_csv("BX-Users.csv")
```

Python

```
new_users['age'].fillna(users['age'].mean(), inplace=True)
```

Python

Αρχικά φορτώνουμε τις απαραίτητες βιβλιοθήκες καθώς και το csv file με τους users σ' ένα καινούργιο dataframe στο οποίο θα μπορούμε και να τροποποιήσουμε τις αρχικές τιμές του αρχείου χωρίς αυτό να επηρεαστεί.

Η πρώτη προ-επεξεργασία των δεδομένων που κάναμε είναι να συμπληρώσουμε όπου δεν υπάρχει ηλικία διαθέσιμη για τον χρήστη με τη μέση ηλικία όλων των χρηστών.

```
new_users['location'] = new_users['location'].str.split(',')
location_info = pd.DataFrame(new_users["location"].tolist()).fillna('').add_prefix('country_')
new_users = pd.concat([new_users, location_info], axis=1)

new_users.drop(['location', 'country_0', 'country_1', 'country_3', 'country_4', 'country_5', 'country_6', 'country_7',
                'country_8'], axis=1, inplace=True)
new_users.rename(columns={"country_2": "country"}, inplace=True)
```

✓ 1.4s

Python

```
for i, place in enumerate(new_users['country']):
    place = place.strip()
    if (place == '' or place == 'n/a'):
        new_users.at[i, 'country'] = 'usa'
    elif (place.isalpha() == False):
        new_users.at[i, 'country'] = 'usa'
    else:
        new_users.at[i, 'country'] = place
```

new\_users

✓ 11.3s

Python

Δεύτερη προ-επεξεργασία που έγινε είναι στο column location του dataframe το οποίο είχε στη γενική περίπτωση 3 locations χωρισμένα με κόμματα, όπου το τελευταίο location ήταν αυτό που μας ενδιέφερε, η χώρα, και στη χειρότερη περίπτωση περισσότερα από 2 κόμματα. Η κάθε τριάδα τοποθεσιών λοιπόν χωρίστηκε με βάση το κόμμα και η κάθε τοποθεσία τοποθετήθηκε μέσα σε μία λίστα όπου αυτή η λίστα αντικατέστησε την προηγούμενη τιμή του πεδίου αυτού.

Στη συνέχεια δημιουργήθηκε ένα νέο dataframe με τις ίδιες εγγραφές που είχε και το dataframe users αλλά το οποίο περιείχε μόνο τις τοποθεσίες, οι οποίες χωρίστηκαν η κάθε μία σε ξεχωριστό column με χαρακτηριστικό μπροστά τη λέξη "country\_" και δίπλα τον αριθμό από 0 κτλ ανάλογα με πόσα στοιχεία είχε η λίστα.

Έπειτα έγιναν merge τα δύο αυτά dataframes με όρισμα axis=1 ώστε να γίνει το merge ως προς τα columns.

Τέλος εφόσον είχε παρατηρηθεί ότι ο μέγιστος αριθμός στοιχείων που μπορούσε να είχε η λίστα ήταν 9 αφαιρέθηκαν τα αχρείαστα columns από το dataframe και πλέον έχουμε μόνο πληροφορίες για το id του χρήστη, την ηλικία και τη χώρα από την οποία κατάγεται.

Στο δεύτερο cell κάνουμε πάλι προ-επεξεργασία για τη χώρα ώστε να ελέγξουμε για τυχόν null τιμές ή μη αλφαριθμητικές και βάζουμε χωρίς βλάβη της γενικότητας ότι αυτά τα άτομα κατάγονται από την Αμερική. Αν δεν υπάρχει λάθος στη χώρα από την οποία κατάγεται ο user την αφήνουμε όπως έχει.

Το αποτέλεσμα που προκύπτει είναι το εξής:

	uid	age	country
0	1	34.565775	usa
1	2	18.000000	usa
2	3	34.565775	ruusia
3	4	17.000000	portugal
4	5	34.565775	usa
...	...	...	...
206442	278853	17.000000	usa
206443	278855	50.000000	usa
206444	278856	34.565775	canada
206445	278857	34.565775	usa
206446	278858	34.565775	ireland
206447 rows × 3 columns			

```
users_enc = pd.get_dummies(new_users,columns=["country"])
users_enc
```

✓ 0.8s Python

	uid	age	country_a	country_aaa	country_aberdeenshire	country_acoruña	country_adsgfdr	country_afghanistan
0	1	34.565775	0	0	0	0	0	0
1	2	18.000000	0	0	0	0	0	0
2	3	34.565775	0	0	0	0	0	0
3	4	17.000000	0	0	0	0	0	0
4	5	34.565775	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
206442	278853	17.000000	0	0	0	0	0	0
206443	278855	50.000000	0	0	0	0	0	0
206444	278856	34.565775	0	0	0	0	0	0
206445	278857	34.565775	0	0	0	0	0	0
206446	278858	34.565775	0	0	0	0	0	0

206447 rows × 587 columns

Στη συνέχεια στο πάνω cell που φαίνεται στο column country κάνουμε one hot encoding επειδή το machine learning model δέχεται μόνο αριθμητικά δεδομένα οπότε δημιουργούμε τόσα columns όσα τα διαφορετικά locations και βάζουμε την τιμή 1 στο column το οποίο δηλώνει την χώρα από την οποία κατάγεται ο χρήστης και στα υπόλοιπα columns βάζουμε την τιμή 0.

```
kmeans = KMeans(3)
label = kmeans.fit_predict(users_enc.drop(columns='uid'))
cluster_count=Counter(label)
cluster_count
```

✓ 28.8s Python

c:\Python310\lib\site-packages\sklearn\cluster\\_kmeans.py:870: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
warnings.warn(  
Counter({1: 129053, 0: 49658, 2: 27736}))

```
users1=new_users[kmeans.labels_==0]
users2=new_users[kmeans.labels_==1]
users3=new_users[kmeans.labels_==2]
```

✓ 0.3s Python

Στο επόμενο cell δημιουργούμε στιγμιότυπο του μοντέλου KMeans με  $K = 3$  που δηλώνει ότι οι χρήστες μας θα χωριστούν σε 3 ομάδες. Κάνουμε το prediction για να δούμε σε ποιο cluster θα καταταχτεί ο κάθε χρήστης λαμβάνοντας υπόψιν όλα τα columns του one hot encoded dataframe πλην του column uid το οποίο δεν βοηθάει κάπως στο clustering. Έπειτα χρησιμοποιούμε από τη βιβλιοθήκη collections την κλάση Counter για να δημιουργήσουμε ένα Counter object το οποίο θα λάβει ως όρισμα το array που επιστρέφει ο KMeans ώστε να μετρήσει πόσα δείγματα έχει η κάθε ομάδα.

Στο δεύτερο cell φτιάχνουμε 3 νέα dataframes από το new\_users dataframe κι επιλέγουμε με τη σειρά της ομάδας μόνο τους χρήστες που ανήκουν στη συγκεκριμένη ομάδα. Αυτό μπορούμε να το κάνουμε εφόσον δώσαμε το ίδιο dataframe στον KMeans (χωρίς το column id αλλά αυτό δεν έχει σημασία) κι το ίδιο μοντέλο μας επέστρεψε ως array με τη σειρά, σε ποια ομάδα ανήκει η κάθε εγγραφή, δηλαδή user, του dataframe μας.

Για κάθε συστάδα χρηστών φτιάχνουμε ένα dataframe στο οποίο βρίσκονται όλες οι αξιολογήσεις τους

```
ratings1=pd.merge(ratings, users1,on='uid', how='outer', indicator='Exist')
ratings1.drop(['age', 'country'],axis=1, inplace=True)
ratings1=ratings1[(ratings1['Exist']=='both')]
```

✓ 2.2s

```
ratings2=pd.merge(ratings, users2, how='outer', indicator='Exist')
ratings2.drop(['age', 'country'],axis=1, inplace=True)
ratings2=ratings2[(ratings2['Exist']=='both')]
```

✓ 1.5s

```
ratings3=pd.merge(ratings, users3, how='outer', indicator='Exist')
ratings3.drop(['age', 'country'],axis=1, inplace=True)
ratings3=ratings3[(ratings3['Exist']=='both')]
```

✓ 1.2s



Αυτό που αναφέρει το markdown στην κορυφή υλοποιείται καθώς κάνουμε merge, join λειτουργία της sql, τα δύο dataframes των αρχικών βαθμολογιών όλων των χρηστών με τους χρήστες της κάθε ομάδας και κρατάμε μόνο τα uuids που εμφανίζονται και στο ένα dataframe και στο άλλο με την τελευταία εντολή κάθε cell. Έχοντας κάνει drop τα αχρείαστα πλέον columns age και country τα τελικά dataframes βαθμολογιών περιέχουν μόνο τις βαθμολογίες κάθε χρήστη σε βιβλία χωρισμένα ανά ομάδα.

```
ratings1 = ratings1.groupby(['isbn'])['rating'].mean().reset_index(name='average')
books1 = pd.merge(ratings1, books, on='isbn', how='outer')
ratings1 = ratings1[(ratings1['average'] > 0)].sort_values(by='average', ascending=False)
ratings1 = ratings1['isbn'].tolist()
```

✓ 0.4s

```
ratings2 = ratings2.groupby(['isbn'])['rating'].mean().reset_index(name='average')
books2 = pd.merge(ratings2, books, on='isbn', how='outer')
ratings2 = ratings2[(ratings2['average'] > 0)].sort_values(by='average', ascending=False)
ratings2 = ratings2['isbn'].tolist()
```

✓ 0.2s

```
ratings3 = ratings3.groupby(['isbn'])['rating'].mean().reset_index(name='average')
books3 = pd.merge(ratings3, books, on='isbn', how='outer')
ratings3 = ratings3[(ratings3['average'] > 0)].sort_values(by='average', ascending=False)
ratings3 = ratings3['isbn'].tolist()
```

✓ 0.2s

Σε αυτό το κομμάτι κώδικα υπολογίζουμε τη μέση βαθμολογία που έχουν βάλει οι χρήστες της κάθε συστάδας για κάθε βιβλίο που έχουν βαθμολογήσει. Αυτό γίνεται με χρήση της μεθόδου groupby στο column isbn κι έπειτα από το column rating υπολογίζεται η μέση τιμή για το συγκεκριμένο isbn και το column rating πλέον θα ονομάζεται average, διαδικασία η οποία ακολουθείται για κάθε συστάδα. Έπειτα φτιάχνουμε 3 νέα dataframes books για την κάθε συστάδα που περιέχουν τα βιβλία που έχουν βαθμολογήσει ώστε να γίνουν train στη συνέχεια τα νευρωνικά δίκτυα με βάση τα summaries και τις βαθμολογίες.

Στο τέλος όπως και στο προηγούμενο ερώτημα κρατάμε τις βαθμολογίες που είναι μεγαλύτερες από το 0 για κάθε συστάδα και τις ταξινομούμε με φθίνοντα αριθμό και τελικά κρατάμε μόνο τα isbn ως λίστα σε κάθε μεταβλητή rating.

```

user_cluster = 0
goodRatedBooks2 = goodRatedBooks.copy()
if (users1['uid'].isin([user]).any()):
    user_cluster = 1
    goodRatedBooks2.extend(ratings1)
elif (users2['uid'].isin([user]).any()):
    user_cluster = 2
    goodRatedBooks2.extend(ratings2)
elif (users3['uid'].isin([user]).any()):
    user_cluster = 3
    goodRatedBooks2.extend(ratings3)
goodRatedBooks2 = goodRatedBooks2[:1024]

```

✓ 0.0s

Σ' αυτό το κομμάτι του κώδικα αναγνωρίζουμε σε ποιο cluster βρίσκεται ο χρήστης και προσθέτουμε στην ήδη υπάρχουσα λίστα με τα βιβλία που έχει ταξινομήσει τα βιβλία που έχει ταξινομήσει η ομάδα στην οποία βρίσκεται ώστε να του προκύψουν και καινούργιες προτάσεις από τη μηχανή αναζήτησης. Ο μέγιστος αριθμός των βιβλίων που μπορεί να τσεκάρει η elasticsearch είναι 1024 γι' αυτό κρατήσαμε μόνο τόσα.

```

result = books_ex2["hits"]["hits"][:int(0.1*total_hits)]
li = []
for i in result:
    res = i["_source"]
    li.append(res['isbn'])
print(li)

```

✓ 0.0s

```

['0871238829', '0143312146', '0002005018', '0743223527', '1583485473', '0595206263', '0968288308', '1879384493', '0966986105',
'0840734530', '059043893X', '0345391691', '0060973897', '1573225126', '0679310002', '0440225078', '193072229X', '0609605925',
'1563411148', '0945397410', '0140274154', '0345409469', '0140304770', '0345307674', '0394404289', '0380012774', '0743444051',
'0671619055', '0743412273', '0307001385', '0679439242', '0816614024', '1587991039', '0440512158', '0765342294', '0316919896',
'0373871864', '0515107662', '0140286780', '0380810336', '0451456718', '0812533550', '0836220889', '0060923717', '0345421825',
'0590629808', '0590956159', '0671517643', '0312311354', '034536662X', '0786404019', '0446603775', '0553277537', '0399147101',
'0500280673', '0440228204', '0026890380', '0140309578', '0312180624', '1561794708', '1566840287', '0060504080', '1573226106',
'0517576988', '0451201515', '1583940634', '0449212858', '0140344438', '0142000345', '0376026138', '0440226848', '0679824111',
'0743222261', '076422249X', '0802137008', '0399226907', '0671705091', '0399145869', '0688009387', '0486270610', '0679723226',
'0736909672', '0965881199', '0375413278', '0786869011', '0140266909', '0451524063', '0373166931', '0449005844', '0142000671',
'0898861098', '0755200381', '0884270610', '0439313899', '0883681056', '1561840092', '1855381125', '0064472574', '0380003821',

```

Το query που χρησιμοποιήθηκε για να πάρουμε τα βιβλία από την elasticsearch είναι το ίδιο με πριν. Ως αποτέλεσμα βλέπουμε ψηλότερα δύο άλλα βιβλία τα οποία περιέχουν την λέξη Canada και έχουν καλό βαθμό από την ομάδα στην οποία βρίσκεται ο χρήστης. Αυτό είναι πολύ καλό καθώς του έγιναν δύο νέες προτάσεις βιβλίων αν δούμε ότι το πρώτο αποτέλεσμα που υπήρχε πριν ήταν το αποτέλεσμα από το βιβλίο που είχε βαθμολογήσει ο ίδιος και τώρα βρίσκεται στην τρίτη θέση.

### Ερώτημα 3

Στο τρίτο και τελευταίο ερώτημα ζητείται η εκπαίδευση ενός νευρωνικού δικτύου για κάθε συστάδα χρηστών, το οποίο θα προσπαθεί να προβλέψει πως η συγκεκριμένη συστάδα χρηστών θα βαθμολογούσε τα βιβλία που δεν έχουν αξιολογηθεί.

```
✓ from gensim.models import Word2Vec
import gensim
import numpy as np
from keras.layers import Input, Dense
from keras.models import Model, Sequential
import tensorflow as tf
from tensorflow import keras
0.7s
```

Αρχικά φορτώνουμε τις απαραίτητες βιβλιοθήκες.

```
new_books= books[['summary']]
summaries = new_books.summary.apply(gensim.utils.simple_preprocess)
✓ 5.8s
```

Χρησιμοποιούμε το simple\_preprocess της gensim για να κάνουμε προ-επεξεργασία στις περιλήψεις των βιβλίων

```

li = []
i = 0
j = 0
arr = np.empty([100, ], dtype = np.float32)
for review in summaries:
    for word in review:
        try:
            arr += model.wv.get_vector(word)
            j += 1
        except Exception as e:
            arr += np.zeros([100, ])
            i += 1
    li.append(arr)
    arr = np.empty([100, ], dtype = np.float32)

```

✓ 18.7s

Στο πρώτο cell, μέσω της genism, μετατρέπουμε κάθε λέξη σε διάνυσμα με την τεχνική wordtovec. Συγκεκριμένα δημιουργούμε ένα λεξιλόγιο με τις λέξεις των περιλήψεων και στην συνέχεια το εκπαιδεύουμε, έτσι ώστε οι λέξεις να περιγράφονται από διανύσματα.

Στο επόμενο cell δημιουργούμε ένα numpy array για κάθε περίληψη στο οποίο προστίθενται τα διανύσματα των λέξεων που περιλαμβάνει η περίληψη. Στην περίπτωση που συναντήσουμε λέξη που δεν διατίθεται το διάνυσμα της, την προσπερνάμε.

```

new_books1 = books1
new_books1['summary']=li
new_books2 = books2
new_books2['summary']=li
new_books3 = books3
new_books3['summary']=li

```

✓ 0.1s

Στο επόμενο μας βήμα αντιγράφουμε τα dataframes των βιβλίων για τις τρεις συστάδες χρηστών, που είχαμε δημιουργήσει στο δεύτερο ερώτημα, σε τρία νέα dataframes και αντικαθιστούμε στην στήλη 'summary' τις λεκτικές περιλήψεις με τα διανύσματά τους.

```
rated_books1 = new_books1[new_books1['average'].notna()]
unrated_books1 = new_books1[new_books1['average'].isna()]
unrated_books1.reset_index(drop=True, inplace=True)
```

✓ 0.1s

```
rated_books2 = new_books2[new_books2['average'].notna()]
unrated_books2 = new_books2[new_books2['average'].isna()]
unrated_books2.reset_index(drop=True, inplace=True)
```

✓ 0.1s

```
rated_books3 = new_books3[new_books3['average'].notna()]
unrated_books3 = new_books3[new_books3['average'].isna()]
unrated_books3.reset_index(drop=True, inplace=True)
```

✓ 0.1s

Σ' αυτό το κομμάτι του κώδικα ξεχωρίζουμε τα βιβλία που έχουν αξιολογηθεί με αυτά που δεν έχουν για κάθε μια συστάδα. Ο λόγος που κάνουμε αυτόν τον διαχωρισμό είναι ότι θέλουμε να βάλουμε σαν είσοδο στο νευρωνικό τις περιλήψεις των βιβλίων και σαν έξοδο τις αξιολογήσεις τους. Τα βιβλία χωρίς αξιολογήσεις θα εισέλθουν στο ήδη εκπαιδευμένο νευρωνικό δίκτυο για να προβλέψουμε τις βαθμολογίες τους

```

X1 = rated_books1[['summary']]
X1_pred = unrated_books1[['summary']]
y1 = rated_books1['average']

X1 = np.array(X1.summary.tolist()).reshape(-1, len(X1.summary[0]))
X1_pred = np.array(X1_pred.summary.tolist()).reshape(-1, len(X1_pred.summary[0]))
y1 = np.array(y1)

X1_pred[np.isnan(X1_pred)] = 0.0001
X1[np.isnan(X1)] = 0.0001

```

Python

```

m = -1000000000000000000
l = -1000000000000000000
for i in X1:
    if i.max() > m:
        m = i.max()
for i in y1:
    if i.max() > l:
        l = i.max()

X1 = X1 / m

m = -1000000000000000000
l = -1000000000000000000
for i in X1_pred:
    if i.max() > m:
        m = i.max()
for i in y1:
    if i.max() > l:
        l = i.max()

```

Στην συνέχεια του προγράμματος, μετατρέπουμε τα dataframes με τα διανύσματα των περιλήψεων σε δισδιάστατα numpy arrays. Η πρώτη διάσταση είναι ο συνολικός αριθμός των διανυσμάτων, δηλαδή ο συνολικός αριθμός των περιλήψεων, και η δεύτερη η διάσταση του διανύσματος, η οποία είναι 100. Μετά θέτουμε στα nan values των διανυσμάτων μια πάρα πολύ μικρή αριθμητική τιμή, έτσι ώστε να μην μας δημιουργήσουν πρόβλημα στην συνέχεια και κανονικοποιούμε τα δεδομένα μας. Συγκεκριμένα βρίσκουμε το διάνυσμα με την μέγιστη τιμή και διαιρούμε κάθε διάνυσμα με αυτό.

```

X2 = rated_books2[['summary']]
X2_pred = unrated_books2[['summary']]
y2 = rated_books2['average']

X2 = np.array(X2.summary.tolist()).reshape(-1, len(X2.summary[0]))
X2_pred = np.array(X2_pred.summary.tolist()).reshape(-1, len(X2_pred.summary[0]))
y2 = np.array(y2)

X2_pred[np.isnan(X2_pred)] = 0.0001
X2[np.isnan(X2)] = 0.0001

```

Python

```

m = -1000000000000000000
l = -1000000000000000000
for i in X2:
    if i.max() > m:
        m = i.max()
for i in y2:
    if i.max() > l:
        l = i.max()

X2 = X2 / m

m = -1000000000000000000
l = -1000000000000000000
for i in X2_pred:
    if i.max() > m:
        m = i.max()
for i in y2:
    if i.max() > l:
        l = i.max()

```

```

X3 = rated_books3[['summary']]
X3_pred = unrated_books3[['summary']]
y3 = rated_books3['average']

X3 = np.array(X3.summary.tolist()).reshape(-1, len(X3.summary[0]))
X3_pred = np.array(X3_pred.summary.tolist()).reshape(-1, len(X3_pred.summary[0]))
y3 = np.array(y3)

X3_pred[np.isnan(X3_pred)] = 0.0001
X3[np.isnan(X3)] = 0.0001

```

Python

```

m = -1000000000000000000
l = -1000000000000000000
for i in X3:
    if i.max() > m:
        m = i.max()
for i in y3:
    if i.max() > l:
        l = i.max()

X3 = X3 / m

m = -1000000000000000000
l = -1000000000000000000
for i in X3_pred:
    if i.max() > m:
        m = i.max()
for i in y3:
    if i.max() > l:
        l = i.max()

```

Εκτελούμε πανομοιότυπες εντολές και για τις 2 άλλες συστάδες.

```

y1 = np rint(y1).astype(np.int64)
y2 = np rint(y2).astype(np.int64)
y3 = np rint(y3).astype(np.int64)

```

✓ 0.5s



Στο επόμενο μας βήμα χρησιμοποιούμε την εντολή `rint` της `numpy` για να στρογγυλοποιήσουμε τις βαθμολογίες σε ακέραιους αριθμούς από 0 έως 10 με σκοπό κάθε αριθμός να αντιστοιχεί σε μια έξοδο στο νευρωνικό δίκτυο.

```
model1 = keras.Sequential([
    keras.layers.Dense(1000, input_shape=(100, ), activation='relu'),
    keras.layers.Dense(11, activation='softmax')
])

model1.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])

model1.fit(X1, y1, epochs=3)
```

✓ 5.9s Python

Epoch 1/3  
467/467 [=====] - 3s 3ms/step - loss: 2.1043 - accuracy: 0.3153  
Epoch 2/3  
467/467 [=====] - 2s 3ms/step - loss: 2.0842 - accuracy: 0.3155  
Epoch 3/3  
467/467 [=====] - 1s 3ms/step - loss: 2.0811 - accuracy: 0.3163

Έπειτα εκπαιδεύουμε το νευρωνικό μας δίκτυο με τα δεδομένα της πρώτης συστάδας. Χρησιμοποιούμε την συνάρτηση ενεργοποίησης `relu` στην είσοδο και την συνάρτηση ενεργοποίησης `softmax` στην έξοδο. Επιλέγουμε 3 εποχές και τον βελτιστοποιητή `adam`. Το νευρωνικό μας δίκτυο έχει 100 εισόδους, ώστε να υπάρχει αντιστοιχία με την διάσταση του `numpy array` με το διάνυσμα της περίληψης και 11 εξόδους, μια για κάθε βαθμό, όπως προαναφέραμε. Τέλος, στο μεσαίο `layer` εμπεριέχονται 100 νευρώνες.

```
new_average1=model1.predict(X1_pred)
predicted_average1 = [np.argmax(i) for i in new_average1]
unrated_books1['average'] = predicted_average1
```

✓ 11.7s

Χρησιμοποιούμε το εκπαιδευμένο μοντέλο για να εκτιμήσουμε τις βαθμολογίες των μη αξιολογημένων βιβλίων και τοποθετούμε τις προβλεπόμενες βαθμολογίες στο dataframe μας.

Συγκεκριμένα κάθε δίνεται μια πιθανότητα για κάθε έναν από τους βαθμούς 0 έως 10 και ο βαθμός με την υψηλότερη πιθανότητα χρησιμοποιείται ως η εκτιμώμενη βαθμολογία.

Στην περίπτωση μας πρέπει να αναφερθεί ότι ο βαθμός 0 εμφανίζει πάντα την μεγαλύτερη πιθανότητα, καθώς περίπου το 70% των αξιολογήσεων που δίνονται στο αρχικό csv file είναι 0.

```
✓ model2 = keras.Sequential([
    keras.layers.Dense(100, input_shape=(100, ), activation='relu'),
    keras.layers.Dense(11, activation='softmax')
])

✓ model2.compile(optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

model2.fit(X2, y2, epochs=3)

new_average2=model2.predict(X2_pred)
predicted_average2 = [np.argmax(i) for i in new_average2]
unrated_books2['average'] = predicted_average2
```

```

model3 = keras.Sequential([
    keras.layers.Dense(100, input_shape=(100, ), activation='relu'),
    keras.layers.Dense(11, activation='softmax')
])

model3.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])

model3.fit(x3, y3, epochs=3)

```

+ Code

+ Markdown

```

new_average3=model3.predict(x3_pred)
predicted_average3 = [np.argmax(i) for i in new_average3]
unrated_books3['average'] = predicted_average3

```

Εκπαιδεύουμε 2 ακόμα μοντέλα και φορτώνουμε τις προβλεπόμενες βαθμολογίες με τον ίδιο τρόπο που επεξηγήθηκε προηγουμένως για τις υπόλοιπες 2 συστάδες.

```

final_books1 = pd.concat([rated_books1, unrated_books1], axis=0)
final_books1.drop('summary', axis=1, inplace=True)
b1=books1[['isbn', 'summary']]
final_books1= pd.merge(b1, final_books1, on='isbn', how='outer')

```

✓ 0.2s

```

final_books2 = pd.concat([rated_books2, unrated_books2], axis=0)
final_books2.drop('summary', axis=1, inplace=True)
b2=books2[['isbn', 'summary']]
final_books2= pd.merge(b2, final_books2, on='isbn', how='outer')

```

✓ 0.2s

```

final_books3 = pd.concat([rated_books3, unrated_books3], axis=0)
final_books3.drop('summary', axis=1, inplace=True)
b3=books3[['isbn', 'summary']]
final_books3= pd.merge(b3, final_books3, on='isbn', how='outer')

```

✓ 0.2s

Κατόπιν, για κάθε μια από τις συστάδες, ενώνουμε, μέσω της εντολής concat, το dataframe των βιβλίων που είχαν αξιολογηθεί στα προηγούμενα ερωτήματα με το dataframe που έχουμε προσθέσει τα βιβλία με τις αξιολογήσεις που προήλθαν από το νευρωνικό δίκτυο.

Διαγράφουμε την στήλη που εμπεριέχει την περίληψη σε μορφή διανύσματος. Τέλος συνενώνουμε, μέσω της εντολής merge, το παραπάνω dataframe με το dataframe των βιβλίων για που δημιουργήσαμε στο ερώτημα 2 για κάθε συστάδα.

```

goodRatedBooks3 = goodRatedBooks.copy()
if (user_cluster == 1):
    goodRatedBooks3.extend(final_books1)
elif (user_cluster == 2):
    goodRatedBooks3.extend(final_books2)
elif (user_cluster == 3):
    goodRatedBooks3.extend(final_books3)
goodRatedBooks3 = goodRatedBooks3[:1024]

```

✓ 0.1s

Σ' αυτό το κομμάτι του κώδικα ακολουθούμε την ίδια διαδικασία με το 2<sup>ο</sup> ερώτημα. Αναγνωρίζουμε σε ποιο cluster βρίσκεται ο χρήστης και προσθέτουμε στην ήδη υπάρχουσα λίστα με τα βιβλία που έχει αξιολογήσει, τα βιβλία που έχουν αξιολογηθεί από το νευρωνικό δίκτυο για την συστάδα στην οποία βρίσκεται ώστε να του προκύψουν και καινούργιες προτάσεις από τη μηχανή αναζήτησης.

```
result = books_ex3["hits"]["hits"][int(0.1*total_hits)]
li = []
for i in result:
    res = i["_source"]
    li.append(res['isbn'])
print(li)
```

✓ 0.0s Python

['0871238829', '0143312146', '0002005018', '0743223527', '1583485473', '0595206263', '0968288308', '1879384493', '0964778319', '0966986105', '0840734530', '059043893X', '0345391691', '0060973897', '1573225126', '0679310002', '0440225078', '193072229X', '0609605925', '1563411148', '0945397410', '0140274154', '0345409469', '0140304770', '0345307674', '0394404289', '0380012774', '0743444051', '0671619055', '0743412273', '0307001385', '0679439242', '0816614024', '1587991039', '0440512158', '0765342294', '0316919896', '0373871864', '0440168724', '0515107662', '0140286780', '0380810336', '0451456718', '0812533550', '0836220889', '0060923717', '0345421825', '0590629808', '0590956159', '0671517643', '0312311354', '034536662X', '0786404019', '0446603775', '0553277537', '0399147101', '0500280673', '0440228204', '0026890380', '0140309578', '0312180624', '1561794708', '0060504080', '1573226106', '0440183057', '0517576988', '0451201515', '1583940634', '0449212858', '0140344438', '0142000345', '0376026138', '0679824111', '0743222261', '076422249X', '0802137008', '0399226907', '0671705091', '0399145869', '0688009387', '0486270610', '0679723226', '0736909672', '0965881199', '0375413278', '0786869011', '0140266909', '0451524063', '0449005844', '0142000671', '088029261X', '0898861098', '0755200381', '0884270610', '0883681056', '1561840092', '1855381125', '0064472574', '0380003821', '0440414539', '0312985207', '0425122956', '0743454146', '087542791X', '0451525884', '0689710682', '0553577123', '0553279025', '0486268705', '0385333706', '0385720327', '0771079567', '1894294424', '0345326342', '0553205803', '0671625837', '0679734406', '0761121323', '096774590X', '042514545X', '0553568779', '1853260274', '0140043519', '0380978407', '0671881884', '0345403959', '0380718839', '044990542X', '0553096060', '1556618662', '0140446745']

Το query που χρησιμοποιήθηκε για να πάρουμε τα βιβλία από την elasticsearch είναι το ίδιο με πριν. Παρατηρούμε πως η απάντηση που μας επιστρέφει η elasticsearch είναι σχεδόν ίδια με αυτή του δεύτερου ερωτήματος. Βελτίωση δεν παρατηρήθηκε καθώς οι βαθμολογίες για τα μη βαθμολογημένα βιβλία που επέστρεψε το νευρωνικό ήταν κατά κύριο λόγο 0 αφού υπήρχαν πολλές βαθμολογίες 0 και στα αρχικά δεδομένα.